Oasis: An Architecture for Simplified Data Management and Disconnected Operation

Anthony LaMarca, Intel Research Seattle
Maya Rodrig, University of Washington

IRS-TR-03-003

May 2003

Intel Research

Oasis: An Architecture for Simplified Data Management and Disconnected Operation[1]

Anthony LaMarca
Intel Research Seattle
1100 NE 45th Street,
6th Floor
Seattle, WA 98105
lamarca@intel-research.net

Maya Rodrig
University of Washington
Department of Computer Science and Engineering
Seattle, WA 98195-2350
rodrig@cs.washington.edu

Abstract. Oasis is an asymmetric peer-to-peer data management system tailored to the requirements of ubiquitous computing applications and the constraints of ubiquitous computing environments. Drawing upon applications from the literature, we motivate three high-level requirements for ubicomp storage: availability, manageability and programmability. Achieving these properties in the face of heterogeneous devices and frequent disconnections is challenging. Oasis addresses the challenge by distributing weighted replicas and performing background self-tuning. In this paper we describe the Oasis architecture and an initial implementation. Our performance evaluation and implementation of three applications using Oasis suggest that it offers good availability and performance while providing a simple API and a familiar consistency model.

1 Introduction

The vision of ubiquitous computing is an environment in which users, computing and the physical environment are artfully blended to provide insitu interactions that increase productivity and quality of life. Many of the components required to realize this vision are available today: sensor networks, mobile processors and wireless networking have all been drastically improved in recent years. However, there is a dearth of robust, compelling ubiquitous computing applications to run on these new platforms. This is particularly unfortunate since ubiquitous computing appears to be a field for which there is no single "killer app" – rather, ubicomp installations increase in impact and appeal as the number of small coexisting applications grows [35]. Therefore it is imperative to address the obstacles to application growth.

We contend that there are so few ubiquitous computing applications because they are too hard to develop, deploy and manage. Ubiquitous computing is a subset of distributed computing, a domain in which programming is notoriously difficult. A number of factors that are particular to ubicomp scenarios make matters even more

[1] This paper has been submitted for publication.

challenging: devices are resource-challenged and faulty, and devices may be continually arriving and departing. While prototypes of a compelling ubicomp application can be deployed in the lab, it is very difficult to build an implementation that is robust and responsive in a realistic environment.

We argue that the best way to foster ubicomp application development is to provide developers of ubicomp applications with a comprehensive set of software services, in effect an "OS for ubicomp". While there has been work in the area of system software for ubiquitous computing, a number of significant challenges remain [17]. In this paper, we address the challenge of providing ubicomp support for one of the more traditional services, namely the storage and management of persistent data. We chose to start with this service as it is both fundamental and challenging; most applications make use of persistent data, and a dynamic, heterogeneous ubicomp installation presents a daunting environment in which to manage data.

We examined fifteen ubicomp applications described in the literature and we have distilled a common set of requirements that fall in the areas of *availability, manageability,* and *programmability*. Based upon these requirements, we have designed and implemented a data management system called Oasis. We have tested the performance of Oasis and used it to implement multiple ubicomp applications in order to understand how well it satisfies these requirements.

The contributions of this work are twofold. First, we offer an investigation of the data management requirements of ubiquitous computing applications. Second, we propose a new architecture to address these requirements, one based on existing systems and database techniques and algorithms.

The rest of the paper is organized as follows. In section 2 we identify the data management requirements of ubiquitous computing applications and draw specific examples from the literature. Section 3 presents the Oasis architecture and our initial implementation. In section 4 we describe our experience constructing three typical ubicomp applications on top of Oasis. We discuss the performance of the system as measured with the workload of one of our applications in section 5. In sections 6, 7 and 8 we compare Oasis to related work, describe future work, and conclude.

2 Data Management Requirements of Ubicomp Applications

Through a survey of fifteen ubiquitous computing applications published in the literature [1,4,6,7,13,14,16,20,21,22,26,29,34,35], we have identified what we believe are the important data management requirements of these applications. The breadth of applications covered in the survey includes smart home applications, applications for enhancing productivity in the workplace, and Personal Area Network (PAN) applications. The specific requirements can be grouped into three areas: *availability*, *programmability*, and *manageability*.

2.1 Availability

Ubicomp applications are being developed for environments in which people expect devices to function 24 hours a day, 7 days a week. The AwareHome [16] and EasyLiving [7], for example, bring ubicomp to household appliances such as refrigerators, microwaves, and televisions that typically operate with extremely high reliability. For many of these ubicomp devices to function, uninterrupted access to a data repository is needed. A storage solution for ubicomp must ensure data is available in the following conditions:

Data access must be uninterrupted, even in the face of device disconnections and failures. Proposed ubicomp environments use existing computing in the home as part of the computing infrastructure [7,20]. A data management solution should be robust to the failure of some number of these devices. Users should be able to power cycle devices with little consequence; turning off a PC in a smart home should not cause the entire suite of ubicomp applications to cease functioning. The challenge of providing high data availability is increased when applications involve an ad hoc or transient set of devices. The data management system must handle both graceful disconnections and unexpected failures, and data must remain available as devices come and go.

Data may need to be accessed simultaneously in multiple locations even in the presence of network partitions. The majority of ubicomp applications we examined include scenarios that require support for multiple devices accessing the same data in multiple locations. Commonly, these applications call for users to carry small, mobile devices that replicate a portion of the user's home or work data [22,34]. Labscape [4] cites disconnection as uncommon, but would like to support biologists who choose to carry a laptop out of the lab. Finally, some applications involve non-mobile devices sharing data over unreliable channels. The picture frame in the Digital Family Portrait [26], for example, communicates with sensors in the home of a geographically remote family member. The loss of the DSL or modem connection at either house will create a partition. In all of these cases, the application scenarios assume the existence of a coherent data management system that supports disconnected operation.

Data can be accessed from and stored on impoverished devices. Ubiquitous computing applications commonly involve inexpensive, resource-constrained devices used for both accessing and storing data. In PAN applications, for example, impoverished mobile devices frequently play a central role in caching data and moving data between I/O and other computational devices [21,22,34]. Ideally a data management system for ubicomp would accommodate the limitations of resource challenged devices; challenged devices would be able to acts as clients, while data could be stored on fairly modest devices when more powerful devices are not available.

2.2 Manageability

Perhaps the single largest factor keeping ubiquitous computing from becoming a mainstream reality is the complexity of managing the system. We have identified a number of features we think are critical for making a ubicomp data management system practical for deployment with real users.

Technical expertise should be required only in extreme cases. By many accounts the "living room of the future" will have the computational complexity of a backend server room; however there will rarely be an expert to manage it. In many cases only non-technical users are present [26] while in extreme applications like PlantCare [20], there are no users at all. In the spirit of IBM's autonomic computing initiative [38], data management for ubiquitous computing environments should be self-managing to the largest extent possible.

Adjustments to storage capacity should be easy and incremental. Many of the ubicomp systems we examined could most appropriately be labeled as platforms on which many small applications and behaviors are installed [6,7,13]. In such an environment, the data management system should be able to grow incrementally to support changing workloads and capacity needs. It should be easy for new devices to add their resources to the collective storage pool and similarly, the system should be able to handle a decrease in storage capacity due to the permanent departure of a device.

The system should adapt to changes within and across applications. The wide variety of devices and applications in a given deployment mandate an adaptive data management strategy. Rather than having users or developers decide how data should be distributed and indexed, this task should be performed by the data management system itself, informed by the dynamic behavior of the entire collection of applications. Consider the location tracking system that is common to many ubicomp applications [4,7,35]. Its job is to track people and objects and produce a mapping for other applications to use. Some applications use this location data infrequently while other scenarios require tens or hundreds of queries against this data per second. A static configuration always runs the risk of either providing poor performance or over-allocating resources. An adaptive solution, on the other hand, is able to detect the activation of a demanding application and adjust priorities accordingly, ensuring good performance and overall system efficiency.

2.3 Programmability

Just as ubicomp environments are difficult to manage, they are equally challenging to develop applications for. The distributed, dynamic and fault-ridden computing environment is a far cry from the computing platforms in which most software engineers are trained. The value of a ubicomp installation increases with the number of applications and behaviors that is present. With this in mind, we have identified a

number of requirements intended to lower the barrier to entry and make reliable, responsive ubicomp applications easier to develop.

The system should offer rich query facilities. Ubiquitous computing applications often involve large amounts of structured sensor data and frequent searches through this data. A common pattern is an application that takes an action if a threshold value is crossed (e.g., going outdoors triggers a display change [26], low humidity triggers plant watering [20], proximity to a table triggers the migration of a user interface [4]). Data management systems that provide indexing and query facilities vastly reduce the overhead in creating efficient implementations of such behaviors.

The system should offer a familiar consistency model. Some distributed storage systems provide "update-anywhere semantics" [32] in which clients can read from, and write to, any data replica in the system at any time, even when disconnected. Most of these systems provide weak consistency guarantees where applications may see writes to the data occur in a different order than they were written in. These weak guarantees can cause a wide range of unpredictable behaviors that make it very difficult to write reliable applications. We believe that a more familiar, conservative consistency model is appropriate for ubicomp environments, even if it results in a loss of flexibility.

The system should provide a single global view of the data. A ubicomp deployment is a distributed system and as such, a decision must be made as to where persistent data should be stored. This decision ideally takes into account which devices are writing and reading the data and the reliability, capacity and load of the potential locations. Creating too few replicas can result in periods of unavailability, while creating too many wastes resources and can degrade performance. In some cases data placement decisions need to be made by the application developer to achieve particular

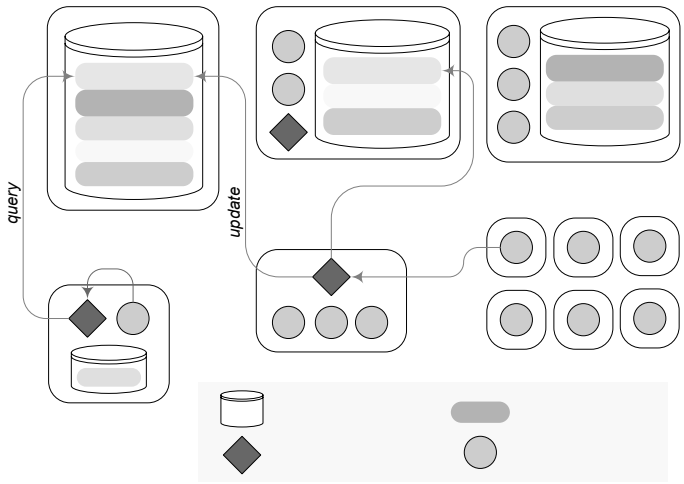

Fig. 1. An example of an Oasis configuration

semantics. Many applications, on the other hand, merely want to reliably store and retrieve data. In these situations, we do not feel that either the user or the application developer should explicitly manage data placement. Accordingly, a data management system for ubicomp should include a facility for making data placement decisions and present the view of a single global storage space to application developers. These decisions can be guided by hints given by the application, but the application should not be directly exposed to a disparate collection of storage devices.

3 The Oasis Architecture

Oasis is a data management system tailored to the requirements of ubiquitous computing. In Oasis, clients access data via a mediator service that in turn communicates with a collection of Oasis servers. The mediator service stores no persistent data; its only purpose is to run the Oasis consistency protocol. The mediator's function has been separated into a distinct service rather than a library routine to allow the participation of extremely impoverished clients (e.g.: sensor beacons that cannot perform two-way communication). Figure 1 shows an example of an Oasis configuration in an instrumented home. Data is replicated across Oasis servers to provide high availability in the event of a device disconnection or failure. Oasis does not depend on any single server; data remains available to clients provided a single replica is available. In the remainder of this section we describe the key elements of the Oasis architecture and describe how these enable Oasis to meet the requirements described in Section 2.

3.1 Data Model

From the client's perspective, Oasis is a database that supports the execution of SQL queries on relational data. We chose SQL because it is a widespread standard for accessing structured data. In addition, SQL offers compact ways to represent complex queries, particularly useful in bandwidth challenged environments. An Oasis installation stores a number of databases. Each database holds a number of tables that in turn hold a set of records. We envision that these databases will collectively contain all of the persistent data used by applications running in the ubiquitous computing environment. Different databases would be created for different types of data such as sensor readings, configuration data, sound samples, etc. In Section 5, we describe three applications we have built using Oasis and their data representation. It should be noted that nothing in the rest of the architecture is specific to the relational data model, and Oasis could be made to manage file- or tuple-oriented data with a few basic changes.

3.2 P2P Architecture with Replication

As devices may arrive and depart in ubiquitous computing scenarios, a distributed architecture that supports dynamic membership is needed. A pure peer-to-peer (P2P) architecture provides the desired decentralization, adaptability, and fault-tolerance by assigning all peers equal roles and responsibilities. However, the emphasis on equal resource contribution by all peers ignores differences in device capabilities. To support a wide variety of devices, Oasis is an asymmetric-P2P system, or "super-peer" system [37], in which devices' responsibilities are based on their capabilities. Devices with greater capabilities contribute more resources and can perform computation on behalf of others, while impoverished devices may have no resources to contribute.

Data is replicated across multiple Oasis Servers to provide high availability. In our initial implementation, replication is done at the database level. (Replicating entire databases simplifies implementation but potentially overburdens small devices. In Section 7 we discuss the potential for partial replication.) An initial replica placement is determined at creation time and is then constantly tuned as devices come and go, and data usage changes. The self tuning process is described in Section 3.4.

3.3 Weighted-Voting and Disconnection Operation

All distributed data-stores employ an access-coordination algorithm that offers a consistency guarantee to the clients accessing the data. To provide developers with a familiar consistency model, we chose an algorithm for Oasis that guarantees clients *sequential consistency* [23] if a local replica is available. Sequential consistency guarantees that the operations of all clients execute in some sequential order, and the operations of each client appear in this total ordering in the same order specified by its

program. Basically, sequential consistency provides a set of distributed clients with the illusion that they are all running on a single device. This is the most natural and easily understood extension of a centralized execution model.

The traditional way to provide sequential consistency and allow disconnected operation is with a quorum-based scheme in which a majority of the replicas must be present to update the data. We have adapted Gifford's "weighted voting" [10] variant of the basic quorum scheme. As in a quorum-based scheme, data replicas are versioned to allow clients to determine which replica is the most recent. In addition, weighted voting assigns every replica of a data object a number of votes. The total number of votes assigned to all replicas of the object is N. A write request much lock a set of replicas whose votes sum to at least W, while read operations must contact a set of replicas whose votes sum to at least R votes. On a read, the client fetches the value from the replica with the highest version number. On a write, the client must update all of the replicas it has locked. Weighted voting ensures sequential consistency by requiring that R+W>N. This constraint guarantees that no read can complete without seeing at least one replica updated by the last write (since R>N-W). Weighted voting is more flexible than a quorum-based approach because the vote allocation as well as R and W can be tailored to the expected workload. Making R small, for example, increases read performance by allowing clients to be serviced by different replicas in parallel. Making R and W close to N/2 allows up to half the servers to fail, increasing fault-tolerance.

Gifford's original weighted-voting algorithm was written for a single, centralized client accessing data on a collection of servers. To allow weighted voting to operate in a P2P system, we developed a decentralized version of Gifford's algorithm. With a single client, the metadata for a replicated data object (R, W, N and the list of replica locations and votes) can be maintained in a centralized fashion at the client's discretion. To enable weighted voting to function in a decentralized manner, we distribute versioned copies of the metadata along with the data in each replica. We allow this metadata to be updated by locking the same votes that are used to update the data and like a data update, a metadata update requires that W votes worth of replicas be locked. This allows data and metadata operations to be safely interleaved, enabling the system to perform self tuning. To guarantee sequential consistency, we add the additional constraint that $W \geq R$. This ensures that both reads and writes of the data see the latest version of the metadata (since $W \geq R > N-W$). With the exception of this additional constraint, our decentralized weighted-voting scheme provides the same flexibility in replica placement and vote assignment as the original. For more detail about our decentralized weighted-voting algorithm see [**Error! Reference source not found.**].

In order to ensure consistency, writes cannot proceed when the required number of votes is not available. In Oasis, SQL queries that attempt to write the database fail in this situation; in effect, a database appears to be "read only" when an insufficient number of votes are available. Queries that only read the data, on the other hand, are permitted to proceed even if a quorum cannot be attained. This is the equivalent of allowing a disconnected client to read from a stale cache of the data. While this may seem to violate sequential consistency, it does not. Since the client cannot acquire a read quorum, they also cannot write the data (since $W \geq R$), ensuring they see a consistent, if out-of-date, snapshot of the data. When Oasis performs a read query on

a potentially stale replica, the query results are marked as stale so that the client can potentially treat the result differently.

Update requests can potentially fail if either the mediator or one of the replicas crashes or departs during the operation. To ensure the consistency of the database, mediators use a two-phase commit protocol when acquiring votes and executing updates on a replica. If a request fails to complete on a replica, the replica will be marked as invalid. Invalid replicas cannot participate in client operations until a distributed recovery algorithm [11] has been successfully executed.

3.4 Online Self-Tuning and Adaptability

Ubiquitous computing applications can generate demanding storage workloads and high variability with respect to the devices they access over time. To meet this challenge without violating our manageability requirements, Oasis has been designed with self tuning in mind. The SQL data model provides the opportunity to add and remove indices. Our weighted voting scheme permits flexibility as to how many replicas are created, where they are placed and what R and W should be. Finally, Oasis ensures that these parameters can be safely updated during a stream of client requests. This allows Oasis to be tuned in an online fashion without denying applications access to the data or requiring user intervention.

Applications have the choice of managing their own replica placement and vote assignment, or allowing Oasis to manage the data on their behalf. For applications that do not want to manage their own replica placement, Oasis includes a self-tuning service that automatically handles replica configuration. When databases are created in Oasis, performance and availability expectations can be provided by applications that want auto-tuning. Oasis servers publish their performance and availability characteristics and the self-tuner uses these along with the application expectations to make its configuration decisions. The self-tuner periodically examines each database's expectations and checks if they are best served by their current replica placement and vote assignment, making adjustments if appropriate. As we discuss in Section 7, we see the development of more sophisticated self-tuning behaviors based on machine learning techniques as a promising direction to pursue for future research.

3.5 Implementation Details

Our initial implementation of Oasis was written in Java. Oasis servers and mediators communicate using an XML-based messaging system called Rain [**Error! Reference source not found.**]. Rain provides a discovery service similar to the ones found in Jini and UPnP. Oasis mediators use this discovery service to find the set of available Oasis servers. Oasis was implemented as a meta-database, meaning that Oasis does not directly store or index data itself, but rather delegates this to an underlying database. The Oasis server has been written to run on top of any JDBC-compliant database that supports transactions. Our initial deployments have used a variety of

products: PostgreSQL and MySql have been used on PC-class devices, and PointBase, an embedded, small-footprint database, has been used with IPAQs and other ARM-based devices. In order to support existing applications that use JDBC, we have written a type-4 JDBC driver for Oasis.

4 Applications

To investigate its usability, we implemented three ubicomp applications on top of Oasis. Two of these applications, the Digital Family Portrait and Dynamo are motivated by applications in the literature while Guide is a new application that has been developed in our laboratory. We did not undertake a rigorous evaluation of our implementations and our conclusions are anecdotal. Having said that, our experience to date suggests that Oasis is a useable, powerful platform that is well suited for the ubicomp domain. More interestingly, for all three applications, we encountered ways in which the capabilities of Oasis transparently augmented or extended some basic function of the application.

4.1 Portrait Display

The Portrait Display is an ongoing project in our laboratory motivated by Mynatt et al.'s Digital Family Portrait [26]. The Digital Family Portrait is part of the Aging in Place project and is aimed at increasing the awareness of extended family members about an elderly relative living alone. Information about the elder's daily life (health, activity level, social interaction) is collected by sensors in his instrumented home, and unobtrusively displayed at the remote home of an extended family member on a digital picture frame surrounding his portrait. The portrait displays both current information as well as trend information over a one week period.

Researchers in our laboratory have been using the digital family portrait scenario to explore various approaches for displaying ambient data about elders that require home care. As part of their investigation, they have developed the Portrait Display, a digital portrait inspired by the original. We have in turn modified this new implementation of the portrait to run on top of Oasis. The four categories of information displayed in our digital portrait are medication intake, meals eaten, outings, and visits. The data used to generate the display comes from a variety of sources. In our prototype, medication and meal information are gathered using Mote-based sensors [12] and cameras. Information about visits and outings is currently entered by hand using a web interface.

The relational data model provided by Oasis is well suited for describing the regular, structured data used by the portrait display application. Similarly, the types of queries needed to extract results to display are easily expressed in SQL. After porting the basic display application, we found the incremental work of adding each subsequent set of sensors to be quite small.

Oasis effectively supports the availability requirements of the portrait display. In an actual deployment, sensors will likely collect data in a variety of locations. Meals,

medications and other activities will be measured in the elder's home, while information about the elder's medical care and social activities could be monitored outside the home. To support this property, the portrait display uses a separate Oasis database for each category of information collected (meals, visits, etc.). Each database is explicitly configured with a replica with 4 votes that resides where data is gathered and a 1-vote replica on the portrait display device (N=5, R=3, W=3). This configuration allows the data to be read and updated at its source, and when a connection exists, allows the portrait display to obtain recent changes. Note that this remains true even if the data source itself is disconnected. For example, after visiting with the elder, a care provider can enter notes about the visit while sitting in his car or back at his office, a practice mentioned in our fellow researcher's interviews with care providers. While the ability to record information when disconnected was not part of the original scenario, the capability is provided by Oasis transparently by placing the 4-vote replica on the care providers laptop or PDA.

This configuration also supports unplanned disconnections by the portrait display itself. Based on the assumption of reliable network connectivity, the original digital family portrait used a simple client-server model in which the display was rendered as a web page fetched from a server running in the elder's home. While suitable for a prototype, it would not work well in a real deployment in which DSL lines and modem connections do in fact go down at times. Implementations that rely on a web client-server model must either display an error page or leave the display unchanged in the case of a disconnection. With Oasis, disconnections are exposed in the form of stale query results giving the application the opportunity to display the uncertainty in an appropriate way.

4.2 Dynamo: Smart Room File System

Stanford's iRoom [13] and MIT's intelligent room [6] are examples of "productivity enhanced workspaces" in which ubiquitous computing is used to help a group of people work more efficiently. In their scenarios, people gather and exchange ideas while sharing and authoring files using a variety of viewing and authoring tools. Generally, in these scenarios either: 1. the files are stored on a machine in the workspace and users lose access when they leave the space, or 2. files are managed by a user's personal device (like a laptop) and everyone else in the workspace loses access when the user departs and disconnects the device.

For our second application we developed a system called Dynamo that spreads the file management responsibility across multiple devices and enables richer usage scenarios. Dynamo allows file-oriented data to be consistently replicated across devices and presents an API that supports use on a standard desktop computer. In Dynamo, each user or group owns a hierarchical tree of directories and files, much like a home directory. Users can choose particular contexts in which to share various portions of their file system with other users (example contexts are 'Oasis code review' or 'hiring meeting'). The collective sum of files shared by the users that are present make up the files available in the workspace. In this manner, everyone present

at a hiring meeting, for example, can share their proxies and interview notes with the other participants without exposing other parts of their file space.

Dynamo was written as an extension to Apache's webDav server. Rather than using a file system, Dynamo stores a user's files in an Oasis database. Microsoft's Web Folders can be used to expose a webDav server as file systems in Windows allowing Dynamo's file hierarchy to be accessed using standard desktop applications. Implementing Dynamo on top of Oasis required a small number of changes to the original webDav server (less than 400 lines). Despite this, the relational data model was not a terribly good fit for the file oriented data stored in Dynamo. Mapping the hierarchy of the file system into relations required a translation step not needed in our other two applications.

The flexibility of Oasis enabled a variety of semantically interesting configurations. If desired, Dynamo can create a 1-vote replica of a user's files on a device that resides in the workspace. This permits the user to disconnect and leave while enabling the remaining participants to open and view (but not write) the files that have been shared. These stale read-only files remain in that context until the user returns at which time a more up to date, writeable version would be seen. For files owned by a group, interesting ownership policies can be arranged by assigning votes based on the user's roles and responsibilities. This can be used to enforce policies ranging from simple majority schemes in which all replicas have equal votes to more complex specifications such as: 'the budget cannot be edited unless the boss plus any other two employees are present'. While this flexibility raises a number of privacy and interface challenges, it shows how Oasis can add rich semantics to a simple application.

4.3 Guide

The Guide project [**Error! Reference source not found.**] aims to exploit, for the purpose of context inference, recent trends towards pervasive deployment of passive Radio Frequency Identification (RFID) tags. The project involves tagging hundreds to thousands of objects in a work space with RFID tags and tracking their position using RF antennas mounted on a mobile sensing platform. Tagged objects include books, personal electronics, office equipment (from staplers to copiers), kitchen appliances and laboratory equipment. As the platform moves around the environment the antennas pick up the ID numbers of nearby tags. For each tag ID *i* discovered at time *t* and location *l* by antenna *a*, the platform writes the tuple *(i, t, l, a)* to a database. The database thus accumulates the location of objects over time. The goal of Guide is to determine high-level relationships between objects based on the accumulated data. For instance, when combined with data on the location of people in the space over a long period, the system may determine that certain objects belong to certain people with high likelihood. On the other hand, the system may examine readings over the last few seconds to detect objects currently in use.

Guide was implemented on top of Oasis. The relational data model was an ideal match for Guide's structured rfid readings and all of the Guide queries could be easily

expressed as SQL statements. The indexing provided by the underlying database was essential in reducing the time to process Guide queries.

Guide has demanding performance, reliability and availability requirements. First it is expected to generate large quantities of data; in the current configuration, the database is expected to grow to contain millions of entries in months. Given that the Guide database is intended to be used as a common utility by all those in the workspace, it is quite possible that tens or hundreds of clients will query the guide database. The database must therefore scale to support large numbers of potentially complex queries in parallel. Second, this large quantity of data must be stored reliably. Since the data may represent months of activity (and it is impossible to regenerate the data), and the entire period may well be relevant, losing the data will be detrimental. Third, since the queries may be part of standing tasks (such as context-triggered notification) it is important that the database be highly available. Based on Guide's expectations of high availability and performance, the Oasis self-tuner configured the Guide database with three 1-vote replicas (N=3, R=2, W=2). This configuration provides high reliability, good performance and continuous access to the data provided that any two of the three servers are available.

5 Performance

To measure the performance of Oasis using a realistic ubicomp data set and workloads, we constructed an experiment based on the database from the Guide application described in the previous section. The Guide database is comprised of 3 tables: a *reading* table tracking when and where an rfid-tag was seen, an *object* table that relates rfid-tags to object names (like 'stapler'), and finally a *place* table that records the geometric bounds of rooms and locations. Our experimental data set was seeded with 1 million records in the reading table, 1000 records in the object table and 25 records in the place table. This approximates the number of tagged objects in our laboratory and the number of readings we expect to record in a month.

In our benchmark a set of clients alternate between performing queries and updates on the database. The queries are all of the form "Where was object X last seen". These are fairly compute intensive queries that join across the reading and place tables. The updates are inserts of a new record into the reading table. The ratio of queries to updates performed by the clients is 50:1, again approximating the expected workload in a Guide deployment. To show the tradeoffs offered by Oasis, we measure two Oasis configurations: one which offers the highest query performance (R=1, W=N) and another which offers the highest tolerance to server failure (R=N/2, W=N/2+1). The high-performance configuration cannot tolerate any server failures, while in the fault-tolerant configuration, up to $\lceil N/2 \rceil - 1$ replicas can fail or disconnect without affecting clients' access to the data. To show the overhead that Oasis introduces, we compare its performance to direct accesses to the underlying PostgreSQL database. In our experiments, the number of clients is fixed at 10 and the number of replicas is varied from 1 to 6. Each client and server in the test ran on its own Pentium 4 PC running either Windows 2000 or Linux connected via 100MB/s

Ethernet. The Oasis servers and clients ran on Sun's 1.3.1 JVM and the underlying data was stored in PostgreSQL 7.3.

Figure 2 shows the total throughput achieved by the set of clients. The graph shows that for a singly-replicated database, Oasis achieves lower throughput than PostgreSQL. This is expected since Oasis incurs overhead sending messages and managing locking and versioning data. The graph shows that as replicas are added, read queries are able to take advantage of the increased parallelism each new server offers. This parallelism is greater in the high-performance configuration in which a read query can be fully serviced by any one replica. In the fault-tolerant configuration, read queries are slower for two reasons. First, reads incur higher overhead as more votes need to be acquired, and second, there are fewer up to date replicas to read from, reducing parallelism. Nonetheless, sufficient parallelism exists to overcome the overhead introduced by the Oasis protocols. Thus for all multiple-replica configurations, Oasis achieves higher throughput than direct access to a single PostgreSQL server. The variability in throughput for the fault-tolerant configuration is due to two factors. First, as the third replica is added, an optimization can no longer be applied, increasing the messaging overhead. Second, extra servers are available for reading with every other replica added, creating a stair-step effect from that point on.

Figure 3 shows a latency breakdown for the Guide queries executed against a 2-way replicated Oasis database. The total latency time is broken down into three parts, the execution time in the underlying database and the messaging and locking overhead introduced by Oasis. The breakdown of costs shows that read queries spend more time executing in the database than the writes. It also shows that the Oasis overhead is substantially higher for the writes than the reads. With two replicas, read operations send fewer messages than the writes due to an opportunity to piggy-back

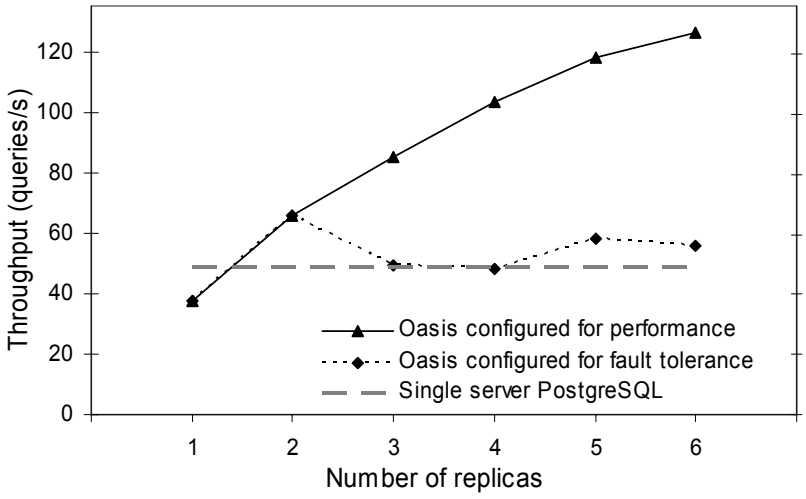

Fig. 2. This graph compares the throughput of two Oasis configurations and a single PostgreSQL server. In the experiment ten clients are running the Guide workload concurrently

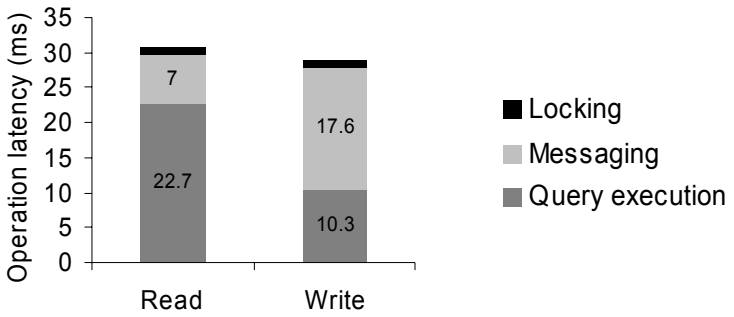

Fig. 3. This graph shows the latency breakdown of read and write queries in the Guide workload for Oasis configured with two replicas.

the query on the lock request. This figure illustrates that the majority of the Oasis overhead is spent in Rain's XML-based messaging system. This suggests that using an optimized message transport could offer substantial performance gains.

6 Related Work

There are many existing storage management systems available to ubicomp application developers, including file systems, databases, and tuple-stores. Distributed variants of these exist which provide clients with uninterrupted access to data despite arrival and departure of servers [30,31,33]. These distributed systems exhibit a variety of behaviors when clients disconnect from the network. In most systems, disconnected clients are unable to read or write data, others offer limited disconnected operation [27], while some systems give clients full read and write facilities while disconnected [15,18,32]. We now review the storage management systems that are most relevant to Oasis and discuss how they compare.

A number of data management systems permit clients to perform updates to a local replica at any time, even when the device on which the client is running is disconnected from all other replicas of the data. These so called "update anywhere" systems are attractive because they never deny the client application the ability to write data and guarantee that the update will eventually be propagated to the other replicas. There are update-anywhere file-systems such as Coda [18] as well as update-anywhere databases such as Bayou [32] and Deno [15]. As data can be updated in multiple locations at the same time, these systems offer weaker consistency guarantees than Oasis. To achieve eventual consistency, update anywhere systems employ varying mechanism to resolve conflicts that arise between replicas. Coda [18] relies on the user to merge write conflicts that cannot be trivially resolved by the system. This technique is a poor fit for ubicomp environments where the user may not be near a computer to provide input or may not have the necessary level of expertise. In Bayou [32], writes are accompanied by fragments of code that travel with the write request and are consulted to resolve conflicts. While these migrating, application-specific conflict resolvers are a potentially powerful model, we believe that writing

them is far beyond the technical abilities of an average software engineer. Finally, Deno [15] uses rollback to resolve conflicts between replicas. Rollback is not suitable in computing environment like ubicomp that involve actuating devices in the physical world.

The recent popularity of P2P file sharing has given rise to a number of P2P data management systems. While file sharing systems like Gnutella satisfy a number of our requirements they do not provide a single consistent view of the data as servers connect and disconnect. Systems like Farsite [5], OceanStore [19], and CFS [9] improve on the basic P2P architecture by incorporating replication to probabilistically ensure a single consistent view. While these systems share our goals of availability and manageability, there are significant differences that make them less than ideal for ubicomp environments. Farsite was designed for a network of PCs running desktop applications. OceanStore is geared for global-scale deployment and depends on a set of trusted servers. Finally, CFS provides read-only access to clients and is not intended as a general-purpose file system.

A few storage systems have been designed specifically for ubiquitous computing environments. The TinyDB system [25] allows queries to be routed and distributed within a network of impoverished sensor nodes. While TinyDB is restricted to run on TinyOS sensor networks, their goals for availability and manageability largely overlap ours. Systems like PalmOS allow PDA users to synchronize their data with a desktop computer. These require explicit action on the part of the user and do not scale beyond a few devices per person. TSpaces [24] and the Event Heap [14] are both tuple-based system that were written for environments with a changing set of heterogeneous devices. These systems are easy to deploy and manage and offer query and indexing facilities. The most significant difference between Oasis and these systems is that their implementations are centralized and ours is not, reflecting our goal to provide high-availability even in the face of disconnection.

Self tuning has been incorporated into several storage systems outside the domain of ubicomp. HP AutoRaid [36] automatically manages the migration of data between two different levels of Raid arrays as access patterns change. Similarly, Hippodrome [3] employs an iterative approach to automating storage system configuration. While these two systems focus on automatically improving the physical layout of data, SQLServer [2] provides an auto-tuning mechanism to improve the selection of indexes and views in the database.

7 Future Work

Our experiences with the current Oasis system have been encouraging. Both performance and ease of use have been surprisingly good. There are however, opportunities to explore and limitations to overcome in future versions of the system.

The largest limitation in Oasis is the need to replicate databases in their entirety. While replicating at the database level simplified our implementation, it potentially places undue burden on application developers. An application that wants to replicate a small part of a large database needs to create an alternate database and keep it consistent. We are considering a variety of ways to solve this problem. One solution

is to change the level at which replication occurs to the table or possibly the record level. Another alternative would be to support partial database replicas similar to 'views' in SQL. We plan to investigate which style of replication best fits ubicomp as well as understand how our protocols would have to change to remain correct.

We are also interested in applying machine learning techniques to the tuning of distributed storage systems like Oasis. We believe that large gains in both availability and query performance can be attained by taking greater advantage of device characteristics and data access patterns. While work has been done in off-line self tuning, little on-line tuning work has been done for dynamic storage systems such as Oasis. We plan to investigate how machine learning techniques can be used to guide the placement of replicas, the creation of indexes and the adjustment of the weighted-voting parameters.

8 Conclusions

It is difficult to write responsive and robust ubiquitous computing applications using traditional data management systems. This is a major hindrance to the ubicomp domain in which success is dependent on the existence of many small applications. To help alleviate this burden, we have built Oasis, a data management system tailored to the characteristics of ubiquitous computing environments. Oasis presents an SQL interface and a relational data model, both of which are well suited to the data usage of typical ubicomp applications. A peer-to-peer architecture coupled with a weighted-voting scheme provides sequentially consistent access to data while tolerating device disconnections and failures. Automatic data placement and background self-tuning allows data to be efficiently managed across a dynamic set of heterogeneous devices using a simple API. We have validated our initial implementation by showing it exhibits good performance as well as using Oasis to implement three typical ubiquitous computing applications.

References

1. Abowd, G. D., Atkeson, C. G., Feinstein, A., Hmelo, C., Kooper, R., Long, S., Sawhney, N., and Tani, M. Teaching and learning as multimedia authoring: the classroom 2000 project. Proceedings of ACM Multimedia '96, 187-198, 1996.
2. Agrawal S., Chaudhuri S. and Narasayya V., Automated Selection of Materialized Views and Indexes for SQL Databases. Proceedings of the 26th International Conference on Very Large Databases, 496-505, 2000.
3. Anderson, E., Hobbs, M., Keeton, K., Spence, S., Uysal, M., and Veitch, A. Hippodrome: running circles around storage administration. In Conference on File and Storage Technology. USENIX, 2002.
4. Arnstein, L., Sigurdsson, S. and Franza, R., Ubiquitous computing in the biology laboratory. Journal of Laboratory Automation, March 2001.
5. Bolosky, W., Douceur J., Ely, D. and Theimer M., Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs", In Proceedings of ACM Sigmetrics, 2000.

6. Brooks, R., The Intelligent Room Project. Proceedings of the Second International Cognitive Technology Conference, 1997.
7. Brumitt, B., Meyers, B., Krumm, J., Kern, A., and Shafer, S. EasyLiving: Technologies for intelligent environments. In Proc. of 2nd International Symposium on Handheld and Ubiquitous Computing (2000), 12-29.
8. Card, S. K., Robertson, G. G., and Mackinlay, J. D.. The information visualizer: An information workspace. Proc. ACM CHI'91 Conf. (1991), 181-188.
9. Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. Wide-area cooperative storage with CFS. In Proceedings of the 18th ACM Symposium on Operating Systems Principles, 2001.
10. Gifford, D. K., Weighted Voting for Replicated Data, Proceedings of the Seventh Symposium on Operating Systems Principles, 1979, pp. 150-162.
11. Goodman, N., Skeen, D., Chan, A., Dayal, U., Fox, S, and Ries, D., A recovery algorithm for a distributed database system, in Proceedings 2nd ACM Symposium on Principles of Database Systems, March, 1983.
12. Hill, J., Szewcyk, R., Woo, A., Culler, D., Hollar, S. and Pister, K.. 2000. System Architecture Directions for Networked Sensors. Architectural Support for Programming Languages and Operating Systems 2000.
13. Johanson B., Fox A. and Winograd T. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. IEEE Pervasive Computing Magazine 1(2), April-June 2002.
14. Johanson, B. and Fox, A., The Event Heap: An Coordination Infrastructure for Interactive Workspaces, Proc. 4th IEEE Workshop Mobile Computer Systems and Applications.
15. Keleher, P., Decentralized Replicated-Object Protocols. In Proc. 18th ACM Symp. on Principles of Distributed Computing, (1999), 143-151.
16. Kidd, C., Orr, R., Abowd, G.D., Atkeson, C.G., Essa, I.A., MacIntyre, B., Mynatt, E., Starner, T.E., and Newstetter, W.: The Aware Home: A Living Laboratory for Ubiquitous Computing Research. Proceedings of the Second International Workshop on Cooperative Buildings, 1999.
17. Kindberg T. and Fox A., System Software for Ubiquitous Computing. IEEE Pervasive Computing, 1(1), Jan 2002, pp. 70-81.
18. Kistler, J., Satyanarayanan, M. Disconnected Operation in the Coda File System. ACM Transactions on Computer Systems, Feb. 1992.
19. Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B., OceanStore: An Architecture for Global-Scale Persistent Storage, ASPLOS, 2000.
20. LaMarca, A., Brunette, W., Koizumi D., Lease M., Sigurdsson S., Sikorski K., Fox D., Borriello G., PlantCare: An Investigation in Practical Ubiquitous Systems. Ubicomp 2002: 316-332
21. Lamming, M. and Flynn, M., Forget-me-not: Intimate Computing in Support of Human Memory, in Proceedings of International Symposium on Next Generation Human Interface, (1994).
22. Lamming, M., Eldridge, M., Flynn M., Jones C., and Pendlebury, D., Satchel: providing access to any document, any time, anywhere, ACM Transactions on Computer-Human Interaction, (7)3:322-352, 2000.
23. Lamport, L. How to make a multiprocessor computer that correctly executes multiprocessor programs. IEEE Trans. on Computers, 28(9):690-691, Sept. 1979.
24. Lehman, T. J, McLaughry, S. W., Wyckoff, P., Tspaces: The next wave. Hawaii Intl. Conf. on System Sciences (HICSS-32), January 1999.
25. Madden S., Franklin M., Hellerstein J., and Hong W., The Design of an Acquisitional Query Processor for Sensor Networks. To Appear, SIGMOD, June 2003.

26. Mynatt, E., Rowan, J., Craighill, S. and Jacobs, A. Digital family portraits: Providing peace of mind for extended family members. Proc of the ACM Conference on Human Factors in Computing Systems, 2001, 333-340.
27. Oracle9i Lite Developers Guide for Windows CE, Release 5.0.1, Jan 2002.
28. Patterson D., Brown A., Broadwell B., Candea G., Chen M., Cutler J., Enriquez P., Fox A., Kiciman E., Merzbacher M., Oppenheimer D., Sastry N., Tetzlaff W., Treuhaft N. Recovery oriented computing (ROC): Motivation, definition, techniques, and case studies. UC Berkeley CS Tech report CSD-02-1175, March 2002.
29. Sumi, Y. and Mase, K, Digital System for Supporting Conference Participants: An Attempt to Combine Mobile, Ubiquitous and Web Computing. Ubicomp 2001.
30. Sweeney, A., Doucette, D., Hu, W., Anderson, C., Nishimoto, M., and Peck, G. Scalability in the XFS File System. In Proc. of the 1996 Winter USENIX, 1996, 1-14.
31. Sybase® Adaptive Server® Enterprise 12.5 A Technical White Paper, http://www.sybase.com/content/1013017/ASE125techwhitepaper.pdf
32. Terry, D., Theimer, M., Petersen, K., Demers, A., Spreitzer, M. and Hauser, C. "Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System", Proc. 15th ACM Symp on Operating Systems Principles, (1995), 172-183.
33. Thekkath, C., T. Mann, and E. Lee. Frangipani: A Scalable Distributed File System. In 16th ACM Symposium on Operating Systems Principles, 1997, 224-237.
34. Want, R., Pering, T., Danneels G., Kumar M., Sundar, M., Light, J.: The Personal Server: Changing the Way We Think about Ubiquitous Computing. Ubicomp 2002: 194-209.
35. Weiser, M., The computer for the twenty-first century. Scientific American, pages 94-100, September 1991.
36. Wilkes, J., Golding, R., Staelin, C., and Sullivan, T. The HP AutoRAID Hierarchical Storage System. ACM Transactions on Computer Systems, 14(1), Feb 1996.
37. Yang, B., and Garcia-Molina, H. Designing a super-peer network. Technical Report, Stanford University, February 2002.
38. Autonomic Computing Manifesto, http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf, visited March 2003.
39. Rodrig M., LaMarca, A. Decentralized Weighted Voting for P2P Data Management, Intel Research Seattle Technical Report IRS-TR-03-004, May 2003.
40. http://seattleweb.intel-research.net/projects/rain/
41. http://seattleweb.intel-research.net/projects/guide/